**Step 6.1 - Begin with the list of the tables that the entities and relationships from the E-R diagram mapped to naturally, from the sample project section at the end of chapter 4.**

For each table on the list, identify functional dependencies and normalize the relation to BCNF. Then decide whether the resulting tables should be implemented in that form. If not, explain why.

The following tables resulted from the mapping:

**Artist** (firstName, lastName, street, city, state, zip, interviewDate, interviewerName, areaCode, telephoneNumber, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

**PotentialCustomer** (firstName, lastName, areaCode, telephoneNumber, street, city, state, zip, dateFilledIn, *preferredArtistLastName, preferredArtistFirstName*, preferredMedium, preferredStyle, preferredType)

**Artwork** (*artistLastName, artistFirstName*, workTitle, askingPrice, dateListed, dateReturned, dateShown, status, workMedium, workSize, workStyle, workType, workYearCompleted, *collectorSocialSecurityNumber*)

**ShownIn** (*artistLastName, artistFirstName, workTitle, showTitle*)

**Collector** (SocialSecurityNumber, firstName, lastName, street, city, state, zip, interviewDate, interviewerName, areaCode, telephonenumber, salesLastYear, salesYearToDate, *collectionArtistFirstName, collectionArtistLastName*, collectionMedium, collectionStyle, collectionType, SalesLastYear, SalesYearToDate)

**Show** (showTitle, *showFeaturedArtistLastName, showFeaturedArtistFirstName*, showClosingDate, showTheme, showOpeningDate)

**Sale** (InvoiceNumber, *artistLastName, artistFirstName*, *workTitle,* amountRemittedToOwner, saleDate, salePrice, saleSalesPersonCommission, saleTax, SaleTotal, *buyerLastName, buyerFirstName, buyerAreaCode, buyerTelephoneNumber, salespersonSocialSecurityNumber*)

**Buyer** (firstName, lastName, street, city, state, zip, areaCode, telephoneNumber, purchasesLastYear, purchasesYearToDate)

**Salesperson** (socialSecurityNumber, firstName, lastName, street, city, state, zip)

**ARTIST**

**Artist** (firstName, lastName, street, city, state, zip, interviewDate, interviewerName, areaCode, telephoneNumber, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

For the Artist table, let us identify FDs.

firstName + lastName → all attributes

It would appear that

socialSecurityNumber → all attributes

Recall that BCNF permits determinants that are candidate keys to remain in the table, so we do not have a problem with leaving socialSecurityNumber in the table. However, we assumed that we would not always have the social security number of artists whose works were owned by collectors, so the value of this attribute may be null for some artists. However, when it appears, it is unique in the table.

zip → city, state

We might want to consider

areaCode →? city, state

We conclude that with mobile telephones, the area code is not necessarily associated with the city and state of the artist’s residence or studio, so we do not have this as a functional dependency.

We also consider whether the complete telephone number determines the address

areaCode + telephoneNumber →? street, city, state, zip

Is it possible for two artist records with the same telephone number to have two different addresses? If the telephone is a landline, the addresses should be the same (the address of the location of the telephone). If it is a mobile telephone, the two artists would have to be sharing it for the same number to appear in two different records. We will keep this as a functional dependency. We might then want to ask

areaCode + telephoneNumber →? all attributes

We decide this is not the case, since we consider the possibility that two artists may share the same home or studio and the same telephone number there, but still have different names, styles, and so on.

The table is in 1NF and 2NF, but not 3NF or BCNF because of the dependencies we have listed. We therefore decompose the Artist table as follows

**Artist1** (firstName, lastName, interviewDate, interviewerName, *areaCode, telephoneNumber*, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

Phones(areaCode, telephoneNumber, street, *zip*)

Zips(zip, city, state)

However, this design would require that we use the telephone number in order to get the street and zip code of an artist, and that we do two joins whenever we want to get an artist’s complete address. For the sake of efficiency, we will compromise and put the street and zip back in the Artist1 table. We choose to leave the Zips table as it is, noting that complete zip code tables are available for purchase in electronic form. Now our form for the Artist tables is

**Artist2** (firstName, lastName, interviewDate, interviewerName, areaCode, telephoneNumber, street, *zip*, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

Zips(zip, city, state)

In practice, it is better to use a numeric field for a key value than a character string, which is subject to differences in spelling, capitalization and punctuation in entering data. Data entry errors or variations in entering string data can cause errors when we try to compare values; in particular, when the fields are used as foreign keys. Therefore, we will create a unique numeric identifier for each artist, which we will call artistId, and make that the primary key of the first table. All the attributes of the first table will be functionally dependent on it. This type of key is sometimes called a **surrogate key**, and most database management systems have a mechanism to generate values and to keep track of values for surrogate keys. Oracle uses identity columns or sequences for this purpose. Microsoft Access uses an **autonumber** data type for the same purpose. We therefore have as our final artist tables:

(1) **Artist3** (artistId, firstName, lastName, interviewDate, interviewerName, *areaCode, telephoneNumber*, street, *zip*, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

(2) **Zips** (zip, city, state)

For PotentialCustomer we have many of the same FDs that we saw for Artist

firstName + lastName + areaCode + telephoneNumber→ all attributes

areaCode + telephoneNumber → street, city, state, zip

zip → city, state

**POTENTIAL CUSTOMER**

**PotentialCustomer** (firstName, lastName, areaCode, telephoneNumber, street, city, state, zip, dateFilledIn, *preferredArtistLastName, preferredArtistFirstName*, preferredMedium, preferredStyle, preferredType)

Using the same reasoning as we did for Artist, we will add a unique identifier, potentialCustomerId, to use as the primary key. We might question whether there is a functional dependency between the preferred artist and other preferences. Although there may be some connection, it is not a true functional dependency, since, for example, two people who admire the same artist might prefer works that are in different mediums or styles, perhaps even produced by the same artist.

For strict BCNF we would break up PotentialCustomer as follows:

Customer1(potentialCustomerId, firstname, lastName, *areaCode, telephoneNumber*, dateFilledIn, *preferredArtistLastName, preferredArtistFirstName*, preferredMedium, preferredStyle, preferredType)

Phones(areaCode, telephoneNumber, street, *zip*)

**Zips** (zip, city, state)

Using the same logic that we used for artist information, we choose instead to create a table that keeps the street and zip code with the other customer data, and to use the Zips table that already exists to determine city and state. We also want to use the preferred artistId instead of first and last name. Therefore, we add to the design the table

(3) **PotentialCustomer2** (potentialCustomerId, firstname, lastName, areaCode, telephoneNumber, street, *zip,* dateFilledIn, *preferredArtistId,* preferredMedium, preferredStyle, preferredType)

**ARTWORK**

**Artwork** (*artistLastName, artistFirstName*, workTitle, askingPrice, dateListed, dateReturned, dateShown, status, workMedium, workSize, workStyle, workType, workYearCompleted, *collectorSocialSecurityNumber*)

For Artwork, we have the following FDs

artistLastName + artistFirstName + workTitle → all attributes

We were using the artist name as a foreign key here. Since we changed the primary key of Artist to artistId, we will change the foreign key as well, replacing the name by the Id in the Artwork table.

Although there is some logical connection between the dates, there is no functional dependency between them. We will also create a unique identifier for each work, artworkId

(4) **Artwork** (artworkId, *artistId*, workTitle, askingPrice, dateListed, dateReturned, dateShown, status, workMedium, workSize, workStyle, workType, workYearCompleted, *collectorSocialSecurityNumber*)

**SHOWIN**

**ShownIn** (*artistLastName, artistFirstName, workTitle, showTitle*)

For the ShownIn table there are no non-trivial functional dependencies, so we could keep the table in its current form. However, we wish to use artworkId to identify the artwork.

(5) **ShownIn** (*artworkId, showTitle*)

**COLLECTOR**

**Collector** (SocialSecurityNumber, firstName, lastName, street, city, state, zip, interviewDate, interviewerName, areaCode, telephonenumber, salesLastYear, salesYearToDate, *collectionArtistFirstName, collectionArtistLastName*, collectionMedium, collectionStyle, collectionType, SalesLastYear, SalesYearToDate)

In the Collector table, we have the FDs

socialSecurityNumber → all attributes

as well as the FDs we saw previously involving telephone numbers and zip codes. We also want to use artistId in place of the artist’s first and last name. We therefore chose to create the table shown below, and to make use of the existing Zips table.

(6) **Collector1** (socialSecurityNumber, firstName, lastName, street, *zip*, interviewDate, interviewerName, areaCode, telephonenumber, salesLastYear, salesYearToDate, *collectionArtistId*, collectionMedium, collectionStyle, collectionType, SalesLastYear, SalesYearToDate)

**SHOW**

**Show** (showTitle, *showFeaturedArtistLastName, showFeaturedArtistFirstName*, showClosingDate, showTheme, showOpeningDate)

For the Show table, we have the FD

showTitle → all attributes

However, we wish to substitute the artistId for the name, as we did for earlier tables. There may be some connection between the featured artist and the show title, but they are not necessarily functionally dependent. For example, two different shows may both feature the same artist, but their titles will be different. The same is true of the theme and the title. There is also some connection, between the opening date and closing date of the show. If we assumed that only one show can open on a given date, then there would be only one closing date associated with that opening date, and we would have a transitive dependency that we would need to remove. Let us add to the assumptions that more than one show can open at the same time. This is a reasonable assumption if the gallery is large enough. In that case there may be shows that have the same opening date but different closing dates. Given these assumptions, the Show table is already normalized.

(7) **Show** (showTitle, *showFeaturedArtistId,* showClosingDate, showTheme, showOpeningDate)

**BUYER**

**Buyer** (firstName, lastName, street, city, state, zip, areaCode, telephoneNumber, purchasesLastYear, purchasesYearToDate)

For Buyer, we have the FD

firstName + lastName + areaCode +telephoneNumber → all attributes

as well as the FDs involving telephone numbers and zips codes, as we saw earlier for Artist and for Collector. As we did for Artist, we will create a numeric primary key, buyerId. Using the same pattern as for those tables, we design a new Buyer table and make use of the Zips table designed previously.

(8) **Buyer** (buyerId, firstName, lastName, street, zip, areaCode, telephoneNumber, purchasesLastYear, purchasesYearToDate)

For the Sale table, we have the FD

invoiceNumber → all attributes

Since each artwork is sold at most once, we also have

artworkId → all attributes

Since it is permissible to keep a candidate key in the relation, this does not present a problem.

Again, we will substitute the artworkId for the artist name and title, and the buyerId for the buyer name and telephone number as foreign keys.

**SALE**

**Sale** (InvoiceNumber, *artistLastName, artistFirstName*, *workTitle,* amountRemittedToOwner, saleDate, salePrice, saleSalesPersonCommission, saleTax, SaleTotal, *buyerLastName, buyerFirstName, buyerAreaCode, buyerTelephoneNumber, salespersonSocialSecurityNumber*)

If the commission is a constant percentage of the sale price, then we have

salePrice → saleSalesPersonCommission

We do not assume that sale price determines tax, since some buyers, such as non-profit organizations, may be tax-exempt. However, the sale total is just the sum of sale price and tax, so we have

salePrice + tax → saleTotal

Actually, any two of these determine the other. We might think we have the FD between salePrice and amountRemittedToOwner However, it is possible that there may be a delay between the time a work is sold and the time an owner is paid, so we could have two sales with the same sale price and different amounts remitted to the owner, because one has not yet been paid.

Removing the FDs identified here, we form the new table

(9) **Sale1** (InvoiceNumber, *artworkId,* amountRemittedToOwner, saleDate, salePrice, saleTax, *buyerId, salespersonSocialSecurityNumber*)

We could also have the table

Commissions(salePrice, saleSalesPersonCommission)

but we do would not need to store this, since the commission is easily calculated. Similarly, because of the arithmetic relationship among the attributes, we do not need to store the table

Totals(salePrice, saleTax, SaleTotal)

**SALESPERSON**

**Salesperson** (socialSecurityNumber, firstName, lastName, street, city, state, zip)

For Salesperson we have

socialSecurityNumber → all attributes

zip → city, state

Removing the transitive dependency and making use of the existing Zips table, we have

(10) **Salesperson** (socialSecurityNumber, firstName, lastName, street, zip)

The tables in boldface, numbered 1-10, will be used as the final set of tables for the relational design for this database.